Appendix A1

COMBINED DOT DENSITY AND DOT SIZE MODULATION

Zhen He
Charles A Bouman
Qian Lin

10003284
M-8658 US


```
/* amfm_8bit.c file */

/* 8 bits/pixel am/fm halftoning algorithm (with partial doting) */
/* Image length and width are assumed to be multiple of 8 */
/* One row serpentine TDED with suppressing each other dot */
/* The output is a tiff file containing 8bit pwm codes */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "coef.h"
#include "tiff.h"
#include "allocate.h"

void amfm(unsigned int,unsigned int,unsigned char *,unsigned char *,\
        short *,unsigned char **,TDEDPARA *,short *,short *);
void amfm_ed_8bits(unsigned int,unsigned int,unsigned char **,unsigned char**,\
                unsigned char **,TDEDPARA *,short *,short *);
int main(int argc, char ** argv)
{
  int i,j;
  unsigned int height,width;
  FILE * fp;
  struct TIFF_img input_img, output_img, mid;
  time_t first, second;
  TDEDPARA *tdedpara = &TDEDcoeff[0];
  short *dotdensityLUT = &OptDensityLUT[0];
  short *dotsizeLUT = &OptSizeLUT[0];


  if(argc<3) {
      printf("usage: %s input_img.tif output_img.tif\n",argv[0]);
      return 1;
  }

  /* read the input image  */
  if ((fp=fopen(argv[1], "rb"))==NULL) {
```

```
    printf("can not open file %s 1\n",argv[1]);
    exit(2);
  }
  if(read_TIFF(fp,&input_img)) {
    printf("error reading input file\n");
    exit(3);
  }
  fclose(fp);

  if((fp=fopen("dbshalf.tif", "rb"))==NULL) {
    fprintf(stderr, "can not open file: dbshalf.tif\n");
    exit(1);
  }
  if(read_TIFF(fp,&mid))
  {
    fprintf(stderr, "error reading file\n");
    exit(1);
  }
  fclose(fp);

  /* Set variable to do timing of algorithm */
  first = time(NULL);

  /* Modify image width to make sure each strip is multiple of 8 */
  width = floor(input_img.width/8)*8;
  height = floor(input_img.height/8)*8;

  /* Allocate memory for entire fm output image. */
  get_TIFF( &output_img, height, width, 'g' );

  amfm_ed_8bits(height,width,input_img.mono,output_img.mono,\
                mid.mono,tdedpara,dotdensityLUT,dotsizeLUT);

  /* show the run time */
  second = time(NULL);
  fprintf(stdout,"\nFinished AM/FM and writing results.\n");
  fprintf(stdout,"Cum. run time: %f sec.\n",difftime(second,first));

  /* write PWM codes image */
  if( (fp = fopen(argv[2],"wb"))==NULL) {
    printf ("cannot open file %s\n", argv[2]);
    exit(4);
  }

  if(write_TIFF(fp,&output_img)) {
    printf ("\nError writing TIFF file %s\n", argv[2]);
    return 1;
  }
  fclose(fp);

  /* free the space */
  free_TIFF(&(output_img));
  free_TIFF(&(input_img));
  free_TIFF(&(mid));
  fflush(stdout);
  return 0;
}
```

```
void amfm_ed_8bits(
  unsigned int height,         /* Input image height */
  unsigned int width,          /* Input image width */
  unsigned char ** contone_img,      /* Input image [height][width] */
  unsigned char ** token_img, /* Output token image [height][width] */
  unsigned char ** dbs_screen,       /* DBS screen used in thresholding of fm
part */
  TDEDPARA *tdedpara,          /* Tone-dependent error diffusion parameters */
  short *dotdensityLUT,        /* Optimal dot density curve */
  short *dotsizeLUT)           /* Optimal dot size curve */
{
  short *fm_err;
  unsigned int i,j;

  /* initialize first row of fm error buffer */
  srand(1); /* fix the seed */
  fm_err = (short*)malloc(sizeof(short) * (width+2));
  for(j = 0; j<width+2; j++)
    fm_err[j] = (rand()%128-64);  /* initialization */

  /* Process the input image with 2 rows each time */
  for(i=0; i<height; i+=2) {
    if((i%600) == 0) printf("amfm_ed: starting row %d\n", i);
    amfm(width,i,contone_img[i],token_img[i],fm_err,dbs_screen,\
         tdedpara,dotdensityLUT,dotsizeLUT);
  }

  free(fm_err);
  return;

}

/* This subroutine only processes 2 rows */
/* Assume width of image is multiple of 8 */
void amfm(
  unsigned int width,    /* Input image width */
  unsigned int i,        /* ith row */
  unsigned char *img_in,       /* ith row of input image array */
  unsigned char *img_out,      /* ith row of output image array */
  short *fm_err,         /* FM error buffer */
  unsigned char ** dbs_screen,       /* dbs_screen[SCREENHEIGHT][SCREENWIDTH] */
  TDEDPARA *tdedpara,    /* Tone-dependent error diffusion parameters */
  short *dotdensityLUT, /* Optimal dot density curve */
  short *dotsizeLUT)     /* Optimal dot size curve  */

{
  short fm_tmp,thresholding;
  short *fm_err_ptr,*tded_ptr;
  short pixela, pixelb, output;
  int j;
  unsigned char *img_in_ptr, *img_out_ptr, *dbs_pat_rowptr;
  short dotdensity, mod_input, error;
  short W1, W2, W3, W4, T2, DT, e1, e2, e3, e4;
  FILE *fp;

  /*-----------------------------------*/
```

```
/* serpentine even rows                */
/*-------------------------------------*/
    /* initial points */
fm_tmp = 0;
fm_err_ptr = fm_err+1;
img_in_ptr = img_in;
img_out_ptr = img_out;

dbs_pat_rowptr = dbs_screen[(i++)%SCREENHEIGHT]; /* Inverse dbs pattern */

/* Index through pixels in pairs */
for(j = 0; j<width; j=j+2 ) {

  /* First process FM (dot density) for left pixel in pixel pair. */

  /* Get first pixel */
  pixela = *(img_in_ptr++);

  /* Use look-up-table to get dot density */
  dotdensity = dotdensityLUT[pixela];

  /* Compute look-up table entries for tone dependent error diffusion */
  tded_ptr = (short*)(tdedpara + dotdensity);
  T2 = *(tded_ptr++);
  DT = *(tded_ptr++);
  W1 = *(tded_ptr++);
  W2 = *(tded_ptr++);
  W3 = *(tded_ptr++);
  W4 = *tded_ptr;

  /* compute dotdensity modified by diffused error */
  mod_input = dotdensity + *fm_err_ptr;

  /* Threshold modifed dotdensity */
  thresholding = mod_input - (dbs_pat_rowptr[j%SCREENWIDTH] * DT + T2);
  output = (thresholding > 0) ? 255 : 0;

  /* Compute weighted errors */
  error = output - mod_input;
  e1 = (W1 * error)>>8;
  e2 = (W2 * error)>>8;
  e3 = (W3 * error)>>8;
  /*e4 = (W4 * error)>>8;*/
  e4 = error - e1 - e2 -e3;
  /* Diffuse error forward in 1-D error buffer */
  *(--fm_err_ptr) -= e4;
  *(++fm_err_ptr) = fm_tmp - e3;
  *(++fm_err_ptr) -= e1;
  fm_tmp = -e2;

  /* Now process FM (dot density) for right pixel in pixel pair. */
  /* Use same TDED parameters as for left pixel. */

  /* Get second pixel */
  pixelb = *(img_in_ptr++);

  /* Use look-up-table to get dot density */
```

```
    dotdensity = dotdensityLUT[pixelb];

    mod_input = dotdensity + *fm_err_ptr;
    error = - mod_input;  /* suppress dot firing at this pixel */

    e1 = (W1 * error)>>8;
    e2 = (W2 * error)>>8;
    e3 = (W3 * error)>>8;
    /*e4 = (W4 * error)>>8;*/
    e4 = error - e1 - e2 -e3;
    /* Using the tded weights of the left pixel */
    *(--fm_err_ptr) -= e4;
    *(++fm_err_ptr) = fm_tmp - e3;
    *(++fm_err_ptr) -= e1;
    fm_tmp = -e2;

/* Begin section on dot size rendering with partial doting */
    if(output) {
      /* Left pixel */
      *(img_out_ptr++) = (dotsizeLUT[pixela]>>1)+NEWRIGHT;
      /* Right pixel */
      if(dotsizeLUT[pixela] & 1)    /* Take care of quantization error */
         *(img_out_ptr++) = ((dotsizeLUT[pixelb]+1)>>1) + NEWLEFT;
      else
         *(img_out_ptr++) = (dotsizeLUT[pixelb]>>1) + NEWLEFT;
    }
    else {
      *(img_out_ptr++) = NEWRIGHT;
      *(img_out_ptr++) = NEWLEFT;
    }
  }   /* end of ith row */


/*-----------------------------------*/
/* serpentine odd rows               */
/*-----------------------------------*/
  fm_tmp = 0;
  /* Set fm error buffer pointer to the end of fm_err buffer */
  fm_err_ptr = fm_err+ width - 1;  /* offset by 1 */
  img_in_ptr = img_in+width*2-2;
  img_out_ptr = img_out+width*2-1;
  *img_out_ptr = NEWRIGHT;  /* Take care of the last pixel in odd row */
  img_out_ptr -= 2;

  dbs_pat_rowptr = dbs_screen[i%SCREENHEIGHT];

  /* Index through pixels in pairs */
  for(j = width-2; j> 0; j=j-2) {
  /* First process FM (dot density) for right pixel in pixel pair */

  /* Get right pixel */
    pixela = *(img_in_ptr--);

    /* Use look-up-table to get dot density */
    dotdensity = dotdensityLUT[pixela];

    /* Compute look-up table entries for tone dependent error diffusion */
```

```
    tded_ptr = (short*)(tdedpara + dotdensity);

    T2 = *(tded_ptr++);
    DT = *(tded_ptr++);
    W1 = *(tded_ptr++);
    W2 = *(tded_ptr++);
    W3 = *(tded_ptr++);
    W4 = *tded_ptr;

    /* Compute dotdensity modified by diffused error */
    mod_input = dotdensity + *fm_err_ptr;

    /* suppress this dot and compute the error */
    error = - mod_input;

    /* Compute weighted errors */
    e1 = (W1 * error)>>8;
    e2 = (W2 * error)>>8;
    e3 = (W3 * error)>>8;
    /*e4 = ((W4 * error)>>8);*/
    e4 = error - e1 - e2 -e3;

    /* duffuse error forward in 1-D error buffer */
    *(++fm_err_ptr) -= e4;
    *(--fm_err_ptr) = fm_tmp - e3;
    *(--fm_err_ptr) -= e1;
    fm_tmp = -e2;

    /* Now process FM (dot density) for Left pixel in a pair */

    /* Get second pixel */
    pixelb = *(img_in_ptr--);

    /* Use look-up-table to get dot density */
    dotdensity = dotdensityLUT[pixelb];

    mod_input = dotdensity + *fm_err_ptr;

    /* Threshold modifed dotdensity */
    thresholding = mod_input - (dbs_pat_rowptr[(j-1)%SCREENWIDTH] * DT + T2);
    output = (thresholding > 0) ? 255 : 0;

    error = output - mod_input;

    e1 = (W1 * error)>>8;
    e2 = (W2 * error)>>8;
    e3 = (W3 * error)>>8;
    /*e4 = (W4 * error)>>8;*/
    e4 = error - e1 - e2 -e3;

    *(++fm_err_ptr) -= e4;
    *(--fm_err_ptr) = fm_tmp - e3;
    *(--fm_err_ptr) -= e1;
    fm_tmp = -e2;

/* Begin section on dot size rendering with partial doting */
    if(output) {
```

```
      /* Left pixel */
      *(img_out_ptr++) = (dotsizeLUT[pixelb]>>1) + NEWRIGHT;
      /* Right pixel */
      if(dotsizeLUT[pixelb] & 1)     /* Take care of quantization error */
         *img_out_ptr = ((dotsizeLUT[pixela]+1)>>1) + NEWLEFT;
      else
         *img_out_ptr = (dotsizeLUT[pixela]>>1) + NEWLEFT;
    }
    else {
      *(img_out_ptr++) = NEWRIGHT;
      *img_out_ptr = NEWLEFT;
    }
    img_out_ptr -= 3;
  }
  *(++img_out_ptr) = NEWLEFT; /* Take care of the first column */
  return;
}
```

Appendix A2

COMBINED DOT DENSITY AND DOT SIZE MODULATION

Zhen He
Charles A Bouman
Qian Lin

10003284
M-8658 US


```
/* coef.h file */

#define SCREENHEIGHT 128
#define SCREENWIDTH  128

#define NEWRIGHT  0xc0
#define NEWLEFT   0x40
#define NEWCENTER 0x00

#define F1 0x0007    /* Floyd-Steinberg Weights 7/16 in Q4 */
#define F2 0x0003    /* Floyd-Steinberg Weights 3/16 in Q4 */
#define F3 0x0005    /* Floyd-Steinberg Weights 5/16 in Q4 */
#define F4 0x0001    /* Floyd-Steinberg Weights 7/16 in Q4 */

typedef struct TDEDPARA
{
      short T2;
      short DT;
      short W1;
      short W2;
      short W3;
      short W4;
} TDEDPARA;


static TDEDPARA TDEDcoeff[129]={
{76, 0, 181, 0, 3, 72},
{76, 0, 181, 0, 3, 72},
{79, 0, 172, 1, 2, 81},
{80, 0, 161, 14, 18, 63},
{82, 0, 159, 1, 37, 59},
{83, 0, 149, 6, 5, 96},
{83, 0, 141, 30, 0, 85},
{85, 0, 138, 13, 0, 105},
{86, 0, 144, 10, 1, 101},
{85, 0, 129, 48, 3, 76},
{86, 0, 123, 31, 1, 101},
{87, 0, 123, 29, 3, 101},
```

```
{87, 0, 115, 28, 5, 108},
{89, 0, 138, 19, 18, 81},
{89, 0, 111, 17, 51, 77},
{88, 0, 115, 31, 0, 110},
{87, 0, 120, 16, 16, 104},
{88, 0, 139, 12, 0, 105},
{89, 0, 122, 19, 17, 98},
{90, 0, 112, 32, 0, 112},
{91, 0, 98, 34, 20, 104},
{90, 10, 123, 16, 26, 91},
{93, 8, 126, 1, 74, 55},
{89, 10, 89, 26, 71, 70},
{89, 10, 89, 22, 43, 102},
{89, 12, 91, 21, 34, 110},
{88, 12, 85, 24, 30, 117},
{88, 14, 85, 23, 30, 118},
{84, 24, 113, 27, 13, 103},
{82, 26, 113, 33, 0, 110},
{83, 26, 109, 29, 9, 109},
{84, 28, 106, 21, 29, 100},
{85, 28, 103, 13, 56, 84},
{96, 2, 102, 16, 57, 81},
{93, 6, 102, 25, 28, 101},
{91, 12, 102, 24, 32, 98},
{96, 2, 103, 24, 23, 106},
{94, 10, 99, 17, 62, 78},
{95, 6, 110, 12, 110, 24},
{97, 4, 114, 12, 112, 18},
{97, 6, 114, 11, 113, 18},
{96, 8, 111, 14, 110, 21},
{94, 12, 102, 17, 109, 28},
{94, 8, 79, 32, 108, 37},
{95, 6, 74, 35, 110, 37},
{97, 2, 70, 35, 111, 40},
{97, 4, 68, 33, 112, 43},
{97, 6, 69, 28, 112, 47},
{98, 6, 70, 22, 114, 50},
{97, 6, 68, 43, 113, 32},
{100, 4, 68, 22, 114, 52},
{99, 6, 71, 24, 112, 49},
{102, 2, 70, 23, 114, 49},
{100, 6, 68, 23, 114, 51},
{100, 8, 66, 22, 116, 52},
{100, 8, 66, 24, 116, 50},
{96, 16, 75, 0, 122, 59},
{95, 16, 63, 0, 127, 66},
{95, 16, 56, 0, 130, 70},
{97, 14, 56, 0, 132, 68},
{97, 16, 59, 0, 132, 65},
{97, 16, 60, 0, 133, 63},
{98, 16, 62, 0, 133, 61},
{95, 26, 98, 0, 109, 49},
{97, 20, 65, 0, 132, 59},
{98, 18, 61, 0, 132, 63},
{99, 18, 63, 0, 131, 62},
{100, 16, 58, 0, 133, 65},
{100, 16, 58, 0, 131, 67},
```

```
{101, 16, 60, 0, 131, 65},
{101, 16, 63, 0, 129, 64},
{101, 16, 58, 0, 129, 69},
{102, 16, 71, 0, 123, 62},
{103, 8, 68, 23, 114, 51},
{103, 8, 66, 22, 116, 52},
{105, 6, 68, 22, 115, 51},
{106, 4, 70, 22, 114, 50},
{108, 2, 69, 23, 113, 51},
{105, 8, 68, 22, 114, 52},
{108, 6, 70, 20, 115, 51},
{106, 8, 69, 27, 112, 48},
{109, 2, 65, 35, 112, 44},
{110, 4, 69, 34, 111, 42},
{110, 6, 72, 35, 110, 39},
{114, 0, 73, 34, 111, 38},
{110, 12, 94, 21, 108, 33},
{111, 12, 102, 15, 110, 29},
{116, 6, 114, 10, 113, 19},
{96, 16, 92, 16, 67, 81},
{100, 12, 95, 17, 67, 77},
{101, 12, 97, 19, 67, 73},
{99, 4, 101, 20, 45, 90},
{93, 4, 103, 25, 25, 103},
{94, 8, 101, 25, 33, 97},
{78, 24, 99, 26, 19, 112},
{81, 26, 104, 22, 24, 106},
{82, 26, 102, 26, 25, 103},
{91, 26, 109, 14, 46, 87},
{104, 10, 82, 0, 95, 79},
{107, 8, 83, 0, 97, 76},
{105, 8, 87, 2, 84, 83},
{81, 14, 86, 27, 25, 118},
{99, 12, 122, 0, 37, 97},
{102, 10, 117, 0, 45, 94},
{103, 10, 90, 21, 64, 81},
{105, 12, 122, 4, 51, 79},
{101, 12, 126, 9, 29, 92},
{88, 12, 121, 25, 0, 110},
{85, 12, 114, 25, 1, 116},
{89, 10, 109, 23, 10, 114},
{86, 12, 112, 29, 1, 114},
{89, 12, 119, 31, 0, 106},
{94, 10, 123, 37, 1, 95},
{93, 8, 117, 63, 1, 75},
{99, 6, 118, 75, 9, 54},
{97, 6, 120, 43, 3, 90},
{111, 6, 121, 35, 32, 68},
{95, 6, 116, 54, 0, 86},
{107, 6, 125, 39, 15, 77},
{93, 34, 137, 27, 19, 73},
{85, 44, 139, 33, 16, 68},
{87, 48, 146, 31, 23, 56},
{87, 44, 148, 22, 10, 76},
{93, 40, 152, 22, 11, 71},
{97, 44, 159, 4, 28, 65},
{95, 42, 161, 25, 4, 66},
```

```
{103, 48, 176, 3, 44, 33},
{101, 56, 165, 27, 55, 9},
{97, 56, 165, 27, 55, 9},
};

static short OptSizeLUT[256]={
120,
118,
117,
115,
114,
112,
111,
109,
108,
106,
105,
104,
102,
101,
100,
99,
97,
96,
95,
94,
93,
92,
91,
90,
90,
89,
88,
87,
86,
86,
85,
84,
84,
83,
82,
82,
81,
81,
80,
80,
79,
79,
78,
78,
77,
77,
77,
76,
76,
76,
75,
```

75,
74,
74,
74,
74,
73,
73,
73,
72,
72,
72,
72,
71,
71,
71,
71,
70,
70,
70,
70,
70,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,

69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,

69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
68,
68,
68,
68,
68,
68,
67,
67,
67,
66,
66,
66,
65,
65,
65,
64,
64,
63,
63,
62,
62,
61,
61,
60,
60,
59,
59,
58,
58,
57,
57,
56,
56,
55,
55,
54,
54,
53,
53,
52,
52,
52,
51,
51,
50,
50,

```
49,
49,
48,
48,
48,
47,
47,
46,
46,
46,
45,
45,
44,
44,
44,
43,
43,
43,
42,
42,
42,
42,
41,
41,
41,
40,
40,
40,
39,
39,
39,
38,
38,
38,
};

static short OptDensityLUT[256]={
128,
127,
126,
125,
124,
123,
122,
121,
120,
119,
119,
118,
117,
117,
116,
115,
115,
114,
114,
113,
```

113,
112,
112,
111,
111,
110,
110,
110,
109,
109,
109,
108,
108,
108,
107,
107,
107,
107,
107,
106,
106,
106,
106,
106,
105,
105,
105,
105,
105,
105,
105,
104,
104,
104,
104,
104,
104,
104,
104,
104,
103,
103,
103,
103,
103,
103,
103,
103,
103,
102,
102,
102,
102,
102,
102,
102,
102,
101,

101,
101,
101,
101,
100,
100,
100,
100,
99,
99,
99,
99,
98,
98,
98,
97,
97,
97,
96,
96,
96,
95,
95,
94,
94,
93,
93,
93,
92,
92,
91,
91,
90,
90,
90,
89,
89,
88,
88,
87,
87,
86,
86,
85,
85,
84,
84,
84,
83,
83,
82,
82,
81,
81,
80,
80,
79,

79,
78,
78,
77,
77,
76,
76,
75,
75,
74,
74,
73,
73,
72,
72,
71,
71,
70,
69,
69,
68,
68,
67,
67,
66,
66,
66,
65,
65,
64,
64,
63,
63,
63,
62,
62,
61,
61,
61,
60,
60,
60,
59,
59,
59,
58,
58,
58,
57,
57,
57,
56,
56,
56,
56,
55,
55,

55,
54,
54,
54,
54,
53,
53,
53,
52,
52,
52,
51,
51,
51,
50,
50,
50,
49,
49,
49,
48,
48,
47,
47,
46,
46,
45,
45,
44,
44,
43,
43,
42,
41,
41,
40,
39,
38,
38,
37,
36,
35,
34,
33,
32,
31,
30,
29,
28,
27,
26,
25,
23,
22,
21,
20,
18,

```
17,
15,
14,
12,
10,
8,
6,
0,
};
```